JDBC - STREAMING ASCII AND BINARY DATA

A PreparedStatement object has the ability to use input and output streams to supply parameter data. This enables you to place entire files into database columns that can hold large values, such as CLOB and BLOB data types.

There are following methods which can be used to stream data:

- **setAsciiStream():** This method is used to supply large ASCII values.
- **setCharacterStream():** This method is used to supply large UNICODE values.
- **setBinaryStream():** This method is used to supply large binary values.

The setXXXStream() method requires an extra parameter, the file size, besides the parameter placeholder. This parameter informs the driver how much data should be sent to the database using the stream.

Example

Consider we want to upload an XML file XML_Data.xml into a database table. Here is the content of this XML file:

```
<?xml version="1.0"?>
<Employee>
<id>100</id>
<first>Zara</first>
<last>Ali</last>
<Salary>10000</Salary>
<Dob>18-08-1978</Dob>
<Employee>
```

Keep this XML file in the same directory where you are going to run this example.

This example would create a database table XML_Data and then file XML_Data.xml would be uploaded into this table.

Copy and past following example in JDBCExample.java, compile and run as follows:

```
// Import required packages
import java.sql.*;
import java.io.*;
import java.util.*;

public class JDBCExample {
   // JDBC driver name and database URL
   static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
   static final String DB_URL = "jdbc:mysql://localhost/EMP";

   //  Database credentials
   static final String USER = "username";
   static final String PASS = "password";

   public static void main(String[] args) {
   Connection conn = null;
   PreparedStatement pstmt = null;
   Statement stmt = null;
   ResultSet rs = null;
   try{
      // Register JDBC driver
      Class.forName("com.mysql.jdbc.Driver");

      // Open a connection
      System.out.println("Connecting to database...");
      conn = DriverManager.getConnection(DB_URL,USER,PASS);
```

```
      //Create a Statement object and build table
      stmt = conn.createStatement();
      createXMLTable(stmt);

      //Open a FileInputStream
      File f = new File("XML_Data.xml");
      long fileLength = f.length();
      FileInputStream fis = new FileInputStream(f);

      //Create PreparedStatement and stream data
      String SQL = "INSERT INTO XML_Data VALUES (?,?)";
      pstmt = conn.prepareStatement(SQL);
      pstmt.setInt(1,100);
      pstmt.setAsciiStream(2,fis,(int)fileLength);
      pstmt.execute();

      //Close input stream
      fis.close();

      // Do a query to get the row
      SQL = "SELECT Data FROM XML_Data WHERE id=100";
      rs = stmt.executeQuery (SQL);
      // Get the first row
      if (rs.next ()){
         //Retrieve data from input stream
         InputStream xmlInputStream = rs.getAsciiStream (1);
         int c;
         ByteArrayOutputStream bos = new ByteArrayOutputStream();
         while (( c = xmlInputStream.read ()) != -1)
            bos.write(c);
         //Print results
         System.out.println(bos.toString());
      }
      // Clean-up environment
      rs.close();
      stmt.close();
      pstmt.close();
      conn.close();
   }catch(SQLException se){
      //Handle errors for JDBC
      se.printStackTrace();
   }catch(Exception e){
      //Handle errors for Class.forName
      e.printStackTrace();
   }finally{
      //finally block used to close resources
      try{
         if(stmt!=null)
            stmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(pstmt!=null)
            pstmt.close();
      }catch(SQLException se2){
      }// nothing we can do
      try{
         if(conn!=null)
            conn.close();
      }catch(SQLException se){
         se.printStackTrace();
      }//end finally try
   }//end try
   System.out.println("Goodbye!");
}//end main

public static void createXMLTable(Statement stmt)
   throws SQLException{
   System.out.println("Creating XML_Data table..." );
   //Create SQL Statement
   String streamingDataSql = "CREATE TABLE XML_Data " +
                             "(id INTEGER, Data LONG)";
```

```
   //Drop table first if it exists.
   try{
      stmt.executeUpdate("DROP TABLE XML_Data");
   }catch(SQLException se){
   }// do nothing
   //Build table.
   stmt.executeUpdate(streamingDataSql);
}//end createXMLTable
}//end JDBCExample
```

Now let us compile above example as follows:

```
C:\>javac JDBCExample.java
C:\>
```

When you run **JDBCExample**, it produces following result:

```
C:\>java JDBCExample
Connecting to database...
Creating XML_Data table...
<?xml version="1.0"?>
<Employee>
<id>100</id>
<first>Zara</first>
<last>Ali</last>
<Salary>10000</Salary>
<Dob>18-08-1978</Dob>
<Employee>
Goodbye!
C:\>
```